Amendments to the Specification

Please replace the paragraph on Page 1, lines 6 - 8 with the following marked-up replacement paragraph:

-- The present invention is a divisional of commonly-assigned U. S. Patent 6,427,161 (serial ~~Patent~~ ________ ~~(serial~~ number 09/097,282, filed on June 12, 1998), which is titled "Thread Scheduling Techniques for Multithreaded ~~titled "Performance Enhancements for Threaded~~ Servers" and which is hereby incorporated herein by reference. --

Please replace the paragraph on Page 8, lines 10 - 20 with the following marked-up replacement paragraph:

-- However, it may be that thread one was nearly finished with its first request at the time the second request arrived. When this is the case, it would be more efficient to wait for the first thread to finish and find the second request when it checks the passive socket, as opposed to awakening the second thread. If the scheduler awakens a new thread for each incoming request (i.e. it over-schedules the threads), a thread working on a request is guaranteed to find the incoming connection queue empty when it completes [[it]] its current request and checks for another. The threads will therefore block after each completed request. The repeated blocking and unblocking operations are expensive in terms of the overall pathlength for servicing a request. When a thread blocks, the scheduler will save the context information for that thread, and the thread will move from the "ready" state to the "blocked" state. The unblocking operation requires the fairly-significant overhead associated with interrupt processing. --

Please replace the paragraph on Page 18, lines 1 - 2 with the following marked-up replacement paragraph:

-- Figures 5A - 5B depict ~~conceptual~~ conceptual representations of the multi-stage queues used by the present invention; and --

Please replace the paragraph that begins on Page 31, line 19 and carries over to Page 32, line 4 with the following marked-up replacement paragraph:

-- The process begins at Step 250, which represents checking for expiration of a "maximum delay" timer. While this is shown as a repeating loop at Step 250, it will be obvious to one of ordinary skill in the art that this checking does not occur uninterrupted. Typically, a timer process of duration "maximum delay" will be scheduled, which causes the timer to begin ticking. Once the maximum delay period has expired, an interrupt will be generated for the timer process, enabling the logic of Steps 255 through 275 to be processed. Alternatively, the checking may be performed more or less often than the maximum delay time, in which case the test in Step 250 would reflect a different time interval. --

Please replace the paragraph on Page 41, lines 1 - 5 with the following marked-up replacement paragraph:

-- Step 610 marks this socket as "Not Ready", and puts an entry for the socket onto the idle connections queue. A counter of the number of idle connections is incremented. An input parameter of the API, the application context value that is associated with this connection, is stored with the socket when it is put onto the idle queue. This enables the application ~~socket~~ to begin processing again, using the same socket, when data subsequently arrives. –

Please replace the paragraph on Page 43, lines 1 - 5 with the following marked-up replacement paragraph:

-- For condition 3, where the connection remains idle too long, the processing of Fig. 6B is used. The delay monitor (see Fig. 3C) that was used for the first and second embodiments, which checked to see if any connections had been on the ready queue too long, is still performed for this third embodiment, as stated above. However, an additional monitoring procedure that checks the idle connections queue is also used, which is illustrated by Fig. 6B. –

Please replace the paragraph on Page 44, lines 1 - 13 with the following marked-up replacement paragraph:

-- Control reaches Step 740 when the connection being checked has been idle beyond the maximum idle time. System resources are not being used efficiently by keeping an ~~association~~ application context open for this connection when it has no data to send, so the connection will be closed down and the resources freed up. Step 740 begins this process, by marking the socket to indicate that it has "Expired", and removing it from the idle queue. The count of idle connections is decremented, and at Step 750, the socket is moved to the ready queue. When the connection is scheduled, the close process will be completed by the worker thread. Steps 760 through 780 perform the scheduling heuristic that was described for the first preferred embodiment, whereby the connection will either be scheduled now by unblocking a thread, or wait to be scheduled when a busy thread becomes available and checks the ready queue of the collector socket (using the modified process of Fig. 3A). When Step 760 indicates that the thread will wait to be scheduled, or Step 780 has finished assigning the connection to an unblocked thread, control returns to Step 720 to see if there are more connections on the idle queue to be checked. –